

**PATENT ABSTRACTS OF JAPAN**

(11)Publication number : 2001-244820

(43)Date of publication of application : 07.09.2001

---

(51)Int.Cl. H03M 13/09  
G06F 11/10  
G06F 12/16  
H04L 1/00

---

(21)Application number : 2000-054468

(71)Applicant : KENWOOD CORP

(22)Date of filing : 29.02.2000

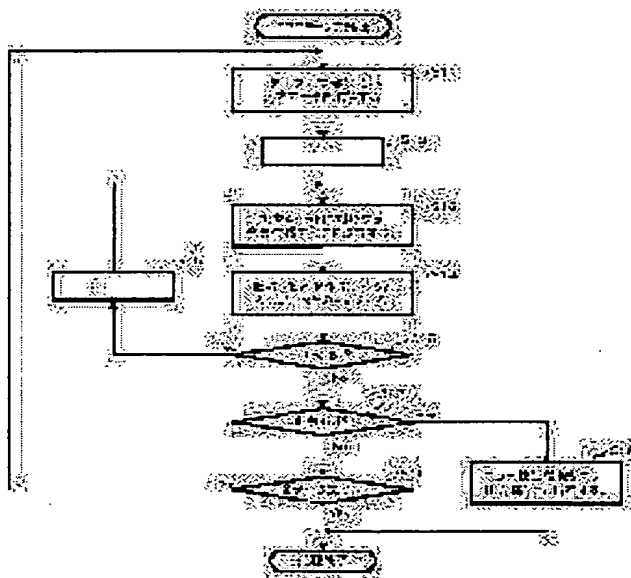
(72)Inventor : HIGUCHI TOMOHISA

---

**(54) ERROR DETECTING DEVICE, CRC CHECK METHOD, AND RECORDING MEDIUM****(57)Abstract:**

**PROBLEM TO BE SOLVED:** To provide an error detecting device which can make a CRC check at a high speed.

**SOLUTION:** An arithmetic procession once obtaining object data from a data input part (step S11) specifies four object bits in the object data and obtains summed data corresponding to the specified four object bits from a generation data table (step S13). The arithmetic procession performs exclusive OR operation between the obtained summed data and input data (step S14). The arithmetic procession repeats the processes of the steps S13 and S14 six times. The arithmetic procession performs exclusive OR operation between the object data and summed data of the six times to decide whether or not residue is generated (step S17). The arithmetic procession when deciding that a residue is generated generates and supplies error detection signal to an error output part (step S18).



---

**LEGAL STATUS**

[Date of request for examination]

12.04.2002

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than  
the examiner's decision of rejection or application  
converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of  
rejection]

[Date of requesting appeal against examiner's  
decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号 ✓  
特開2001-244820  
(P2001-244820A)

(43) 公開日 平成13年9月7日 (2001.9.7)

| (51) Int.Cl. <sup>7</sup> | 識別記号  | F I           | ターミナル* (参考)             |
|---------------------------|-------|---------------|-------------------------|
| H 0 3 M 13/09             |       | H 0 3 M 13/09 | 5 B 0 0 1               |
| G 0 6 F 11/10             | 3 3 0 | G 0 6 F 11/10 | 3 3 0 A 5 B 0 1 8       |
|                           | 3 2 0 |               | 12/16 3 2 0 B 5 J 0 6 5 |
| H 0 4 L 1/00              |       | H 0 4 L 1/00  | A 5 K 0 1 4             |

審査請求 未請求 請求項の数 5 O L (全 13 頁)

(21) 出願番号 特願2000-54468(P2000-54468)

(22) 出願日 平成12年2月29日 (2000.2.29)

(71) 出願人 000003595

株式会社ケンウッド

東京都渋谷区道玄坂1丁目14番6号

(72) 発明者 樋口 知久

東京都渋谷区道玄坂1丁目14番6号 株式会社ケンウッド内

(74) 代理人 100077850

弁理士 芦田 哲仁朗 (外1名)

Fターム(参考) 5B001 AA04 AB01 AC01 AD03 AD06  
AD07

5B018 CA01 HA11

5J065 AB01 AC03 AD08 AE01 AF01

AC01 AH04 AH15 AH18 AH19

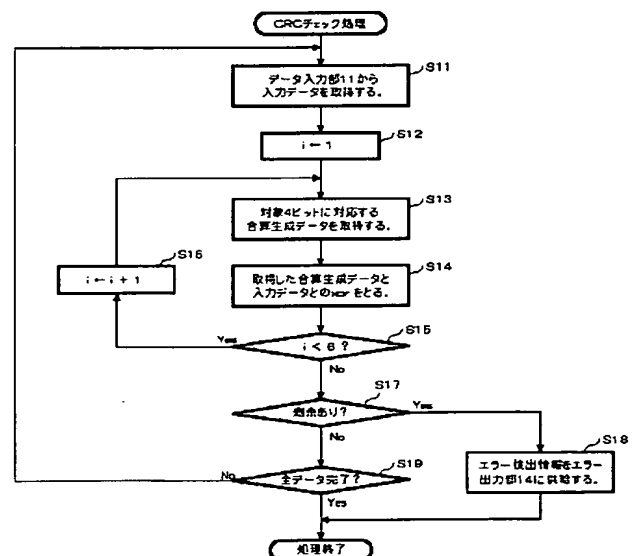
5K014 BA06 EA01 EA02

(54) 【発明の名称】 誤り検出装置、CRCチェック方法及び記録媒体

(57) 【要約】

【課題】 CRCチェックを高速に行うことのできる誤り検出装置を提供することである。

【解決手段】 演算処理部は、データ入力部から対象データを取得すると (ステップ S 1 1)、対象データ中の対象 4 ビットを特定し、特定した対象 4 ビットに対応する合算生成データを生成データテーブルから取得する (ステップ S 1 3)。演算処理部は、取得した合算生成データと入力データとの排他的論理和を演算する (ステップ S 1 4)。演算処理部は、このステップ S 1 3、S 1 4 の処理を 6 回繰り返す。演算処理部は、6 回にわたる対象データと合算生成データとの排他的論理和の演算により、剰余が発生しているか否かを判別する (ステップ S 1 7)。演算処理部は、剰余が発生していると判別した場合、エラー検出情報を生成して、エラー出力部に供給する (ステップ S 1 8)。



## 1

## 【特許請求の範囲】

【請求項 1】 複数ビット分のビット配列に従って、所定の生成多項式が合算されて生成された合算生成データを予め記憶する記憶手段と、

ビット列同士の排他的論理和を演算する演算手段と、

CRC (Cyclic Redundancy Check) チェック対象のデータを入力する入力手段と、

前記入力手段が入力したデータ又は、前記演算手段により前回演算されて生じた演算結果データから複数ビット分のビット配列を特定し、特定したビット配列に対応する合算生成データを前記記憶手段から取得する取得手段と、

前記入力手段が入力したデータ又は、前記演算手段により前回演算されて生じた演算結果データと、前記取得手段が取得した合算生成データとの演算を前記演算手段に所定回数行わせる制御手段と、

前記制御手段の制御により前記演算手段にて所定回数繰り返された演算により生じた演算結果に従って、前記入力手段が入力したデータの誤りを検出するエラー検出手段と、

を備えることを特徴とする誤り検出装置。

【請求項 2】 1 バイト分のビット配列に従って、ビット配置のそれぞれ異なる生成多項式が合算されて生成された合算生成データを予め記憶する記憶手段と、

ビット列同士の排他的論理和を演算する演算手段と、

CRC チェック対象のデータを入力する入力手段と、

前記入力手段が入力したデータ又は、前記演算手段により前回演算されて生じた演算結果データから 1 バイト分のビット配列を特定し、特定したビット配列に対応する合算生成データを前記記憶手段から取得する取得手段と、

前記入力手段が入力したデータ又は、前記演算手段により前回演算されて生じた演算結果データと、前記取得手段が取得した合算生成データとの演算を前記演算手段に所定回数行わせる制御手段と、

前記制御手段の制御により前記演算手段にて所定回数繰り返された演算により生じた演算結果に剰余が含まれる場合に、前記入力手段が入力したデータの誤りを検出するエラー検出手段と、

を備えることを特徴とする誤り検出装置。

【請求項 3】 ビット列同士の排他的論理和を演算する演算ステップと、

CRC チェック対象のデータを入力する入力ステップと、

前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データから複数ビット分のビット配列を特定し、特定したビット配列により定まる生成多項式が合算されて生成された合算生成データを所定の記憶部から取得する取得ステップと、

## 2

前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データと、前記取得ステップにて取得された合算生成データとの演算を前記演算ステップにて所定回数行わせる制御ステップと、

前記制御ステップの制御により前記演算ステップにて所定回数繰り返された演算により生じた演算結果に従って、前記入力ステップにて入力されたデータの誤りを検出するエラー検出ステップと、

10 を備えることを特徴とする CRC チェック方法。

【請求項 4】 ビット列同士の排他的論理和を演算する演算ステップと、

CRC チェック対象のデータを入力する入力ステップと、

前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データから 1 バイト分のビット配列を特定し、特定したビット配列により定まるビット配置のそれぞれ異なる生成多項式が合算されて生成された合算生成データを所定の記憶部から取得する取得ステップと、

20

前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データと、前記取得ステップにて取得された合算生成データとの演算を前記演算ステップにて所定回数行わせる制御ステップと、

前記制御ステップの制御により前記演算ステップにて所定回数繰り返された演算により生じた演算結果に剰余が含まれる場合に、前記入力ステップにて入力されたデータの誤りを検出するエラー検出ステップと、

30

を備えることを特徴とする CRC チェック方法。

【請求項 5】 ビット列同士の排他的論理和を演算する演算ステップと、CRC チェック対象のデータを入力する入力ステップと、前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データから複数ビット分のビット配列を特定し、特定したビット配列により定まる生成多項式が合算されて生成された合算生成データを所定の記憶部から取得する取得ステップと、前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データと、前記取得ステップにて取得された合算生成データとの演算を前記演算ステップにて所定回数行わせる制御ステップと、前記制御ステップの制御により前記演算ステップにて所定回数繰り返された演算により生じた演算結果に従って、前記入力ステップにて入力されたデータの誤りを検出するエラー検出ステップとを有する CRC チェック方法をコンピュータに実行させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体。

40

【発明の詳細な説明】

50 【0001】

## 3

【発明の属する技術分野】本発明は、取得したデータの誤りをCRC (Cyclic Redundancy Check) 方式により検出する誤り検出装置に関し、特にCRCチェックを高速に処理することのできる誤り検出装置、CRCチェック方法及び記録媒体に関する。

## 【0002】

【従来の技術】従来から、送信されたデータや所定の媒体から読み出したデータ等が誤りなく取得できたか否かを判別するために、データの誤り検出が行われている。この誤り検出には、パリティチェック方式やチェックサム方式が用いられてきた。パリティチェック方式は、例えば、送信側にて、7ないし8ビットで表される1文字分のデータに別の1ビットを付加し、全体の「1」のビット数が常に奇数（又は、偶数）個になるようにしてデータを生成する。そして、受信側にて、取得したデータ（1文字分）の「1」のビット数が、奇数（又は、偶数）個でない場合に、エラーが発生したと判断する。また、チェックサム方式は、例えば、送信側にて、一定の長さのブロック内のデータ列を足し合わせた総和を求め、求めた合計値の最下桁1バイトをブロックの最後尾に付加したデータを生成する。そして、受信側にて、取得したデータのブロック内の合計値を求め、合計値の最下桁1バイトと最後尾の1バイトとを比較して、一致しない場合に、エラーが発生したと判断する。

【0003】近年では、これらパリティチェック方式等よりもエラーの検出能力が高く、状況によってはエラーの訂正も可能という点等の理由により、CRC (Cyclic Redundancy Check) 方式が誤り検出に用いられるようになっている。CRC方式は、例えば、送信側にて、ブロック単位のデータ毎に所定の生成多項式を使用して巡回符号を求め、求めた巡回符号をそれぞれ付加したデータを生成する。そして、受信側にて、取得したデータをブロック単位に同一の生成多項式にて除算し、割り切れなかった（剰余が生じた）場合に、エラーが発生したと判断する。

【0004】以下、このCRC方式について、伝送すべきデータをデータD ( $D = X^{23} + X^{21} + X^{18} + X^{17} + X^{14} + X^{13} + X^{12} + X^{11} + X^{10} + X^6 + X^4 + X^3 + 1$ ) とし、生成多項式を多項式G ( $G = X^6 + X^4 + X^3 + 1$ ) とした場合を一例として説明する。送信側は、データDに多項式Gの最高次の項である $X^6$ を乗じたデータDX ( $DX = X^{29} + X^{27} + X^{24} + X^{23} + X^{20} + X^{19} + X^{18} + X^{17} + X^{16} + X^{12} + X^{10} + X^9 + X^6$ ) を多項式Gで割り、その剰余を巡回符号として求める。すなわち、送信側は、図9に示すように、データDXを多項式Gで割り、剰余として、 $X^3 + X^2 + X$ を求める。この $X^3 + X^2 + X$ が巡回符号となる。

【0005】より具体的に説明すると、この場合、データDXは、「1010011001111100010

## 4

11001000000」と表され、また、多項式Gは、「1011001」と表される。そして、データDXを多項式Gで除算する際において、実際には剰余しか必要としないため（商が不要であるため）、送信側は、図10に示すように、データDXと多項式Gとの排他的論理和 (xor) を順次演算し、剰余として「1110」（すなわち、 $X^3 + X^2 + X$ ）を求める。送信側は、この剰余、つまり巡回符号をデータDXに付加したデータ（「101001100111110001011001001110」）を生成して受信側に向けて送信する。

【0006】受信側は、受信したデータを同一の多項式Gで除算し、剰余が生じた場合に、エラーが発生したと判断する。すなわち、受信側は、図11に示すように、受信したデータ（「101001100111110001011001001110」）と多項式G（「1011001」）との排他的論理和を順次演算して、剰余の有無を判別する。このように、受信したデータと多項式Gとを使用した演算により剰余の有無を判別し、受信データの誤り検出を精度良く行うことができる。

## 【0007】

【発明が解決しようとする課題】しかし、上述の図10や図11を参照して説明したように、多項式Gによるデータの除算は、チェック対象データ等の上位ビットが「1」の場合に、ビット配置を合わせた多項式Gとの排他的論理和を演算し、また、「0」の場合に、シフトさせて次のビットを参照するという演算処理を繰り返し行う必要がある。すなわち、ビット単位で演算処理を行っており、排他的論理和等の演算を数多く行う必要があるため、誤り検出のための処理負荷が大きくなり、全体の処理効率の低下を招いていた。

【0008】また、例えば、CD-ROM内に記録したデータ等を読み出す際には、1セクタ分のデータ量である2060バイトものデータに対して、誤り検出を行う必要がある。この際、生成多項式には、誤り検出の精度の向上から、より高次の多項式、例えば、多項式G ( $x$ ) =  $X^{32} + X^{31} + X^{16} + X^{15} + X^4 + X^3 + X + 1$  が用いられることとなる。この場合、2060バイトものデータのビット毎に排他的論理和等の演算を繰り返し行うため、処理負荷が極めて大きくなり、誤り検出のために長時間に渡って処理を行わなければならないといった問題があった。

【0009】本発明は、上記実状に鑑みてなされたもので、CRCチェックを高速に行うことのできる誤り検出装置、CRCチェック方法及び記録媒体を提供することを目的とする。

## 【0010】

【課題を解決するための手段】上記目的を達成するため、本発明の第1の観点に係る誤り検出装置は、複数ビット分のビット配列に従って、所定の生成多項式が合算

## 5

されて生成された合算生成データを予め記憶する記憶手段と、ビット列同士の排他的論理和を演算する演算手段と、CRC (Cyclic Redundancy Check) チェック対象のデータを入力する入力手段と、前記入力手段が入力したデータ又は、前記演算手段により前回演算されて生じた演算結果データから複数ビット分のビット配列を特定し、特定したビット配列に対応する合算生成データを前記記憶手段から取得する取得手段と、前記入力手段が入力したデータ又は、前記演算手段により前回演算されて生じた演算結果データと、前記取得手段が取得した合算生成データとの演算を前記演算手段に所定回数行わせる制御手段と、前記制御手段の制御により前記演算手段にて所定回数繰り返された演算により生じた演算結果に従って、前記入力手段が入力したデータの誤りを検出するエラー検出手段と、を備えることを特徴とする。

【0011】この発明によれば、記憶手段は、複数ビット分のビット配列に従って、所定の生成多項式が合算されて生成された合算生成データを予め記憶する。演算手段は、ビット列同士の排他的論理和を演算する。入力手段は、CRC チェック対象のデータを入力する。取得手段は、入力手段が入力したデータ又は、演算手段により前回演算されて生じた演算結果データから複数ビット分のビット配列を特定し、特定したビット配列に対応する合算生成データを記憶手段から取得する。制御手段は、入力手段が入力したデータ又は、演算手段により前回演算されて生じた演算結果データと、取得手段が取得した合算生成データとの演算を演算手段に所定回数行わせる。エラー検出手段は、制御手段の制御により演算手段にて所定回数繰り返された演算により生じた演算結果に従って、入力手段が入力したデータの誤りを検出する。このように、チェック対象データと合算生成データとの演算を、チェック対象データの複数ビット単位に繰り返すことにより、チェック対象データの生成多項式による除算を少ない演算処理にて実現できる。この結果、CRC チェックを高速に行うことができる。

【0012】上記目的を達成するため、本発明の第2の観点に係る誤り検出装置は、1バイト分のビット配列に従って、ビット配置のそれぞれ異なる生成多項式が合算されて生成された合算生成データを予め記憶する記憶手段と、ビット列同士の排他的論理和を演算する演算手段と、CRC チェック対象のデータを入力する入力手段と、前記入力手段が入力したデータ又は、前記演算手段により前回演算されて生じた演算結果データから1バイト分のビット配列を特定し、特定したビット配列に対応する合算生成データを前記記憶手段から取得する取得手段と、前記入力手段が入力したデータ又は、前記演算手段により前回演算されて生じた演算結果データと、前記取得手段が取得した合算生成データとの演算を前記演算手段に所定回数行わせる制御手段と、前記制御手段の制御により前記演算手段にて所定回数繰り返された演算に

## 6

より生じた演算結果に剰余が含まれる場合に、前記入力手段が入力したデータの誤りを検出するエラー検出手段と、を備えることを特徴とする。

【0013】この発明によれば、記憶手段は、1バイト分のビット配列に従って、ビット配置のそれぞれ異なる生成多項式が合算されて生成された合算生成データを予め記憶する。演算手段は、ビット列同士の排他的論理和を演算する。入力手段は、CRC チェック対象のデータを入力する。取得手段は、入力手段が入力したデータ又は、演算手段により前回演算されて生じた演算結果データから1バイト分のビット配列を特定し、特定したビット配列に対応する合算生成データを記憶手段から取得する。制御手段は、入力手段が入力したデータ又は、演算手段により前回演算されて生じた演算結果データと、取得手段が取得した合算生成データとの演算を演算手段に所定回数行わせる。エラー検出手段は、制御手段の制御により演算手段にて所定回数繰り返された演算により生じた演算結果に剰余が含まれる場合に、入力手段が入力したデータの誤りを検出する。このように、チェック対象データと合算生成データとの演算を、チェック対象データの1バイト単位に繰り返すことにより、チェック対象データの生成多項式による除算を少ない演算処理にて実現できる。この結果、CRC チェックを高速に行うことができる。

【0014】上記目的を達成するため、本発明の第3の観点に係るCRC チェック方法は、ビット列同士の排他的論理和を演算する演算ステップと、CRC チェック対象のデータを入力する入力ステップと、前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データから複数ビット分のビット配列を特定し、特定したビット配列により定まる生成多項式が合算されて生成された合算生成データを所定の記憶部から取得する取得ステップと、前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データと、前記取得ステップにて取得された合算生成データとの演算を前記演算ステップにて所定回数行わせる制御ステップと、前記制御ステップの制御により前記演算ステップにて所定回数繰り返された演算により生じた演算結果に従って、前記入力ステップにて入力されたデータの誤りを検出するエラー検出ステップと、を備えることを特徴とする。

【0015】この発明によれば、演算ステップは、ビット列同士の排他的論理和を演算する。入力ステップは、CRC チェック対象のデータを入力する。取得ステップは、入力ステップにて入力されたデータ又は、演算ステップにて前回演算されて生じた演算結果データから複数ビット分のビット配列を特定し、特定したビット配列により定まる生成多項式が合算されて生成された合算生成データを所定の記憶部から取得する。制御ステップは、入力ステップにて入力されたデータ又は、演算ステップ

## 7

にて前回演算されて生じた演算結果データと、取得ステップにて取得された合算生成データとの演算を演算ステップにて所定回数行わせる。エラー検出ステップは、制御ステップの制御により演算ステップにて所定回数繰り返された演算により生じた演算結果に従って、入力ステップにて入力されたデータの誤りを検出する。このように、チェック対象データと合算生成データとの演算を、チェック対象データの複数ビット単位に繰り返すことにより、チェック対象データの生成多項式による除算を少ない演算処理にて実現できる。この結果、CRCチェックを高速に行うことができる。

【0016】上記目的を達成するため、本発明の第4の観点に係るCRCチェック方法は、ビット列同士の排他的論理和を演算する演算ステップと、CRCチェック対象のデータを入力する入力ステップと、前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データから1バイト分のビット配列を特定し、特定したビット配列により定まるビット配置のそれぞれ異なる生成多項式が合算されて生成された合算生成データを所定の記憶部から取得する取得ステップと、前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データと、前記取得ステップにて取得された合算生成データとの演算を前記演算ステップにて所定回数行わせる制御ステップと、前記制御ステップの制御により前記演算ステップにて所定回数繰り返された演算により生じた演算結果に剰余が含まれる場合に、前記入力ステップにて入力されたデータの誤りを検出するエラー検出ステップと、を備えることを特徴とする。

【0017】この発明によれば、演算ステップは、ビット列同士の排他的論理和を演算する。入力ステップは、CRCチェック対象のデータを入力する。取得ステップは、入力ステップにて入力されたデータ又は、演算ステップにて前回演算されて生じた演算結果データから1バイト分のビット配列を特定し、特定したビット配列により定まるビット配置のそれぞれ異なる生成多項式が合算されて生成された合算生成データを所定の記憶部から取得する。制御ステップは、入力ステップにて入力されたデータ又は、演算ステップにて前回演算されて生じた演算結果データと、取得ステップにて取得された合算生成データとの演算を演算ステップにて所定回数行わせる。エラー検出ステップは、制御ステップの制御により演算ステップにて所定回数繰り返された演算により生じた演算結果に剰余が含まれる場合に、入力ステップにて入力されたデータの誤りを検出する。このように、チェック対象データと合算生成データとの演算を、チェック対象データの1バイト単位に繰り返すことにより、チェック対象データの生成多項式による除算を少ない演算処理にて実現できる。この結果、CRCチェックを高速に行うことができる。

## 8

【0018】上記目的を達成するため、本発明の第5の観点に係る記録媒体は、ビット列同士の排他的論理和を演算する演算ステップと、CRCチェック対象のデータを入力する入力ステップと、前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データから複数ビット分のビット配列を特定し、特定したビット配列により定まる生成多項式が合算されて生成された合算生成データを所定の記憶部から取得する取得ステップと、前記入力ステップにて入力されたデータ又は、前記演算ステップにて前回演算されて生じた演算結果データと、前記取得ステップにて取得された合算生成データとの演算を前記演算ステップにて所定回数行わせる制御ステップと、前記制御ステップの制御により前記演算ステップにて所定回数繰り返された演算により生じた演算結果に従って、前記入力ステップにて入力されたデータの誤りを検出するエラー検出ステップとを有するCRCチェック方法をコンピュータに実行させるためのプログラムを記録する。

【0019】

【発明の実施の形態】本発明の実施の形態にかかる誤り検出装置について、以下図面を参照して説明する。

【0020】図1は、この発明の実施の形態に適用される誤り検出装置の一例を示すブロック図である。図示するように、誤り検出装置1は、データ入力部11と、演算処理部12と、記憶部13と、エラー出力部14とを備えて構成される。データ入力部11は、チェック対象となるデータを順次入力し、入力したデータを演算処理部12に供給する。

【0021】演算処理部12は、CPU (Central Processing Unit) 等からなり、データ入力部11から供給されたチェック対象データを演算処理することにより、データエラーが生じているか否かを判別する。具体的に演算処理部12は、後述するCRCチェック処理を実行し、チェック対象データ中の例えば、対象4ビットから定まる合算生成データを使用した演算処理の結果に従ってエラーの有無を判別する。なお、合算生成データについては、後述する。

【0022】記憶部13は、ROM (Read Only Memory) 等からなり、例えば、図2に示すような生成データテーブル131を予め記憶する。生成データテーブル131は、チェック対象データ中の対象4ビットと、合算生成データとの対応関係を規定するテーブルである。すなわち、生成データテーブル131は、16個の対象4ビット(「0000」～「1111」)に対応する合算生成データ(「000000000000」～「1111110101」)を規定するテーブルである。

【0023】この合算生成データは、排他的論理和の演算に使用される生成多項式を対象4ビットのビット配列に従って合算したデータである。なお、生成多項式には、多項式 $G (G = X^6 + X^4 + X^3 + 1)$ が用いられ

る。以下、合算生成データについて、具体的に説明する。

【0024】例えば、チェック対象データにおける対象4ビット（上位4ビット）のビット配列が「1111」である場合、図3に示すように、多項式Gの先頭位置（ビット配置）をずらしたL1、L2及びL4がそれぞれ合算され（排他的論理和が演算され）、ΣLとして合算生成データが求められる。

【0025】ところで従来の除算の場合、例えば図4（a）に示すように、チェック対象データ（「1111・・・」）と多項式G（「1011001」）との排他的論理和を順次演算する。具体的には、まず対象データとL1との排他的論理和を演算し、その演算結果とL2との排他的論理和を演算し、そして、更に演算結果とL

$$\begin{aligned}(D)''' &= (D \text{ xor } L1) \text{ xor } L2 \text{ XOR } L4 \\ &= D \text{ xor } (L1 \text{ xor } L2 \text{ xor } L4)\end{aligned}$$

【0029】数式2に示すように、データDと、多項式Gのビット配置をずらしたL1、L2及びL4の合算（排他的論理和）の演算結果との排他的論理和を演算することにより、数式1と同一となる最終的な演算結果を得ることができる。つまり、図4（b）に示すチェック対象のデータと、L1、L2及びL4が合算されたΣLとの排他的論理和の演算結果（「00000111・・・」）は、図4（a）に示す最終的な演算結果と等しくなる。このことから、数式3が成立することとなる。

【0030】

$$\text{【数3】 } (D)''' = D \text{ xor } \Sigma L$$

【0031】すなわち、排他的論理和に使用される多項式Gを合算して生成されるΣL、すなわち、合算生成データを使用することにより、図4（a）に示すような対象4ビット（「1111」）に対して3回必要であった排他的論理和の演算を、図4（b）に示すように1回にまとめて行うことができる。このような合算生成データが、対象4ビットの16種類のビット配列（「0000」～「1111」）に応じて予め求められ、それぞれのビット配列に対応づけられた生成データテーブル131が記憶部13に記憶されている。

【0032】図1に戻って、エラー出力部14は、演算処理部12の演算処理結果に従って、エラーが検出された場合に、データの誤りを検出したことを示す所定のエラー検出情報を出力する。

【0033】以下、この誤り検出装置1の動作を、図5を参照して説明する。図5は、誤り検出装置1が行うCRCチェック処理を説明するためのフローチャートである。図5に示すCRCチェック処理は、データ入力部11にチェック対象のデータが入力された際に、逐次開始される。

【0034】まず、演算処理部12は、データ入力部11から入力データを取得する（ステップS11）。例えば、演算処理部12は、チェック対象となる30ビット

4との排他的論理和を演算する。すなわち、チェック対象データの対象4ビットが「1111」の場合、データに対して、以下の数式1に示すような演算が行われる。なお、数式1中に示される（D）'は、データDを1回、多項式Gで割った結果を示すものとする。

【0026】

$$\begin{aligned}\text{【数1】 } (D)' &= D \text{ xor } L1 \\ (D)'' &= (D)' \text{ xor } L2 \\ (D)''' &= (D)'' \text{ xor } L4\end{aligned}$$

【0027】この数式1から、（D）'''とDとの関係を導くと、数式2にて示される関係が成立する。

【0028】

【数2】

のデータを取得する。

【0035】演算処理部12は、変数iに初期値の「1」をセットする（ステップS12）。なお、この変数iは、チェック対象データと合算生成データとの排他的論理和の回数をカウントするために使用される。

【0036】演算処理部12は、対象4ビットに対応する合算生成データを取得する（ステップS13）。すなわち、演算処理部12は、チェック対象データ中の対象4ビットを特定し、特定した対象4ビットに対応する合算生成データを記憶部13に記憶された生成データテーブル131から取得する。例えば、演算処理部12は、図6（a）に示すような30ビットのチェック対象データから対象4ビット（「1010」）を特定し、特定した対象4ビットに対応する合算生成データ（「1010010001」）を図2に示す生成データテーブル131から取得する。

【0037】演算処理部12は、取得した合算生成データと入力データとの排他的論理和を演算する（ステップS14）。例えば、演算処理部12は、図6（a）に示すように、チェック対象データと合算生成データ（「1010010010001」）との排他的論理和を演算する。

【0038】演算処理部12は、変数iが「6」より小さいか否かを判別する（ステップS15）。演算処理部12は、変数iが「6」より小さいと判別した場合、変数iに「1」を加算して（ステップS16）、上述のステップS13に処理を戻す。そして、演算処理部12は、前回のステップS14にて演算された結果（演算結果）から次の対象4ビットを特定し、特定した対象4ビットに対応する合算生成データを生成データテーブル131から取得する。更に、演算処理部12は、取得した合算生成データと前回の演算結果との排他的論理和を演算する。このようなステップS13、S14の処理を、演算処理部12は、変数iが「6」となるまで繰り返す。



【0039】すなわち、演算処理部12は、変数*i*が「2」のときに、図6(b)に示すように対象4ビット(「0010」)に対応する合算生成データ(「0010110010」)を生成データテーブル131から取得し、演算結果と合算生成データとの排他的論理和を演算する。また、演算処理部12は、変数*i*が「3」のときに、図6(c)に示すように対象4ビット(「1111」)に対応する合算生成データ(「1111110101」)を取得し、排他的論理和を演算する。以下同様に、演算処理部12は、変数*i*が「4」のときに、図6(d)に示すように排他的論理和を演算し、変数*i*が「5」のときに、図6(e)に示すように排他的論理和を演算し、そして、変数*i*が「6」のときに、図6(f)に示すように排他的論理和を演算する。

【0040】図5に戻って、演算処理部12は、ステップS15にて、変数*i*が「6」以上である(「6」より小さくない)と判別した場合、剰余が発生しているか否かを判別する(ステップS17)。すなわち、上述の6回にわたるチェック対象データ(前回の演算結果)と合算生成データとの排他的論理和の演算により得られた演算結果に、剰余が含まれているか否かを判別する。

【0041】演算処理部12は、剰余が発生していると判別した場合、所定のエラー検出情報を生成して、エラー出力部14に供給する(ステップS18)。演算処理部12は、エラー検出情報をエラー出力部14に供給すると、CRCチェック処理を終了する。

【0042】一方、剰余が発生していないと判別した場合、演算処理部12は、全データのCRCチェックが完了したか否かを判別する(ステップS19)。すなわち、データ入力部11が入力したデータを全て処理したか否かを判別する。

【0043】演算処理部12は、全データの処理が完了していないと判別した場合、ステップS11に処理を戻し、上述のステップS11～S19の処理を繰り返し実行する。一方、全データの処理が完了したと判別した場合、演算処理部12は、CRCチェック処理を終了する。

【0044】このように、チェック対象データと合算生成データとの演算を、チェック対象データの4ビット単位に繰り返すことにより、チェック対象データの生成多項式による除算を少ない演算処理にて実現できる。この結果、CRCチェックを高速に行うことができる。

【0045】次に、上述した誤り検出装置1を構成に含んだ具体的な機器について、CD-ROM装置を一例として説明する。図7は、本発明の他の実施の形態に係るCD-ROM装置の一例を示す模式図である。図示するように、CD-ROM装置100は、誤り検出装置1と、処理制御部2と、制御信号入力部3と、信号処理回路4と、トラッキングスレッドサーボ回路5と、光ピックアップ6と、スピンドルサーボ回路7と、データバッ

ファ8とを備えて構成される。

【0046】誤り検出装置1は、上述の図1に示したように、データ入力部11と、演算処理部12と、記憶部13と、エラー出力部14とからなり、信号処理回路4から供給された例えば、2060バイトのデータを演算処理し、データエラーが生じているか否かを判別する。

【0047】なお、記憶部13には、上述の図2と異なり、図8(a)に示すような生成データテーブル132が予め記憶される。この生成データテーブル132は、チェック対象データ中の対象1バイトと、合算生成データとの対応関係を規定するテーブルである。すなわち、生成データテーブル132は、256個の対象1バイト(「00000000」～「11111111」)に対応する合算生成データを規定するテーブルである。

【0048】この合算生成データは、排他的論理和の演算に使用される生成多項式を対象1バイトのビット配列に従って合算したデータである。なお、生成多項式には、多項式 $G(x) = X^3 + X^2 + X + 1$ が用いられる。これは、チェック対象のデータが2060バイトであることから、誤り検出の精度を向上させるためである。また、対象を1バイトとしたのは、CRCチェック処理をバイト単位に行うことにより、処理効率を更に高めるためである。

【0049】合算生成データについて具体的に説明すると、合算生成データは、例えば、チェック対象データにおける対象1バイト(上位1バイト)のビット配列が「11111111」である場合、図8(b)に示すように、多項式 $G(x)$ の先頭位置(ビット配置)をずらしたL1、L3、L5及びL7がそれぞれ合算され(排他的論理和が演算され)、ΣLとして求められる。同様に他の合算生成データも、対象1バイトのビット配列に応じて予め求められる。このような、合算生成データが、対象1バイトの256種類のビット配列に応じて予め求められ、それぞれのビット配列に対応づけられた生成データテーブル132が記憶部13に記憶されている。

【0050】図7に戻って、処理制御部2は、CPU及び周辺LSI(Large Scale Integration)を含んだ1チップマイコン等からなり、CD-ROM装置100全体を制御する。すなわち、処理制御部2は、信号処理回路4を制御して、信号処理回路4が図示せぬCD-ROMから光ピックアップ6を介して読み出したデータを誤り検出装置1に供給させる。また、処理制御部2は、誤り検出装置1からエラー検出情報を取得すると、信号処理回路4に対してデータの再読み出しを指示する。

【0051】制御信号入力部3は、所定のパーソナルコンピュータ等の機器から送られる制御信号を入力し、処理制御部2に供給する。

【0052】信号処理回路4は、トラッキングスレッドサーボ回路5、光ピックアップ6及び、スピンドルサーボ回路7等を制御する。そして、信号処理回路4は、光

ピックアップ6を介して、CD-ROMに記録されたデータを読み出す。例えば、信号処理回路4は、1セクタ分となる2060バイトのデータをCD-ROMから読み出し、読み出した情報を誤り検出装置1等に供給する。

【0053】トラッキングスレッドサーボ回路5は、光ピックアップ6をCD-ROMの径方向に平行移動させるための図せぬスレッドモータを駆動制御し、光ピックアップ6をCD-ROMの記録面上の所定の位置へ移動させる。

【0054】光ピックアップ6は、所定の波長のレーザ光をCD-ROMの記録面に向けて照射し、その反射光を受けて電気信号に変換する。光ピックアップ6は、変換した電気信号を読み出したデータとして信号処理回路4に供給する。

【0055】スピンドルサーボ回路7は、CD-ROMを搭載する所定のターンテーブルを回転させるための図せぬスピンドルモータを駆動制御し、所定の回転速度で回転駆動させる。

【0056】データバッファ8は、誤り検出装置1にてCRCチェックがなされたデータを信号処理回路4から取得し、所定のパーソナルコンピュータ等に向けて出力する。

【0057】以下、上述のCD-ROM装置100における誤り検出装置1の動作を、簡単に説明する。誤り検出装置1は、信号処理回路4から送られる2060バイトのチェック対象データを取得する。誤り検出装置1は、チェック対象データ中の対象1バイトを特定し、特定した1バイトのビット配列に対応する合算生成データを生成データテーブル132から取得する。誤り検出装置1は、取得した合算生成データとチェック対象データとの排他的論理和を演算する。そして、誤り検出装置1は、前回の演算結果から次の対象1バイトを特定し、特定した対象1バイトのビット配列に対応する合算生成データを生成データテーブル132から取得する。更に、演算処理部12は、取得した合算生成データと前回の演算結果との排他的論理和を演算する。このような演算処理を所定回数繰り返した後に、誤り検出装置1は、最終的に得られた演算結果に、剰余が含まれているか否かを判別する。誤り検出装置1は、剰余が含まれていると判別した場合、処理制御部2にエラー検出情報を出力する。

【0058】このように、チェック対象データと合算生成データとの演算を、チェック対象データの1バイト単位に繰り返すことにより、チェック対象データの生成多項式による除算を少ない演算処理にて実現できる。この結果、CRCチェックを高速に行うことができる。

【0059】なお、上記の実施の形態では、CD-ROM装置等におけるCRCチェックについて説明したが、CRCチェックの対象は任意である。例えば、FDドラ

イブ、メモリ、MPEG変換装置、通信装置、及び、ハードディスク等種々の機器にて行われるCRCチェックに適用可能である。

【0060】なお、この発明の誤り検出装置は、専用のシステムによらず、通常のコンピュータシステムを用いて実現可能である。例えば、コンピュータに上述のいずれかを実行するためのプログラムを格納した媒体（フロッピーディスク、CD-ROM等）から該プログラムをインストールすることにより、上述の処理を実行する誤り検出装置を構成することができる。

【0061】また、コンピュータにプログラムを供給するための手法は、任意である。例えば、通信回線、通信ネットワーク、通信システム等を介して供給してもよい。一例を挙げると、通信ネットワークの掲示板（BBS）に当該プログラムを掲示し、これをネットワークを介して配信する。そして、このプログラムを起動し、OSの制御下で、他のアプリケーションプログラムと同様に実行することにより、上述の処理を実行することができる。

【0062】

【発明の効果】以上説明したように、本発明によれば、CRCチェックを高速に行うことができる。

【図面の簡単な説明】

【図1】本発明の実施の形態に係る誤り検出装置の構成の一例を示すブロック図である。

【図2】記憶部に記憶される生成データテーブルの一例を示す模式図である。

【図3】合算生成データの算出について説明する模式図である。

【図4】（a）が従来の多項式を使用して排他的論理和の演算を順次行う様子を説明するための模式図であり、（b）が合算生成データを使用して排他的論理和の演算を行う様子を説明するための模式図である。

【図5】本発明の実施の形態に係るCRCチェック処理を説明するためのフローチャートである。

【図6】CRCチェック処理により、チェック対象データと合算生成データとの排他的論理和の演算が順次行われる様子を説明するための模式図である。

【図7】誤り検出装置が含まれるCD-ROM装置の構成の一例を説明するための模式図である。

【図8】（a）が記憶部に記憶される生成データテーブルの一例を示す模式図であり、（b）が合算生成データの算出について説明する模式図である。

【図9】生成多項式にてデータが除算される様子を説明するための模式図である。

【図10】従来の巡回符号生成処理において演算される対象データと多項式との排他的論理和について説明するための模式図である。

【図11】従来のCRCチェック処理において演算される対象データと多項式との排他的論理和について説明す

10

20

30

40

50

15

16

るための模式図である。

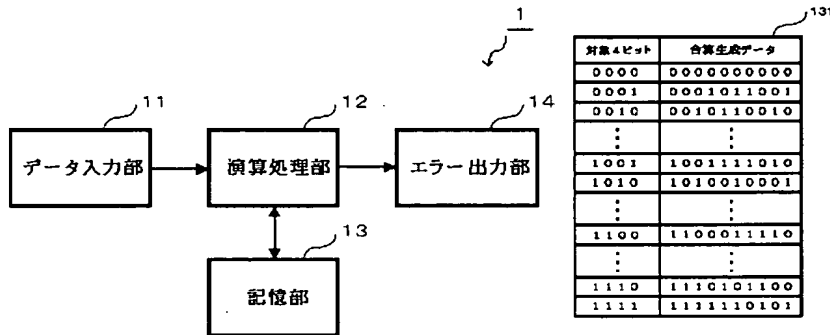
【符号の説明】

- 1 誤り検出装置  
2 処理制御部  
3 制御信号入力部  
4 信号処理回路  
5 トラッキングスレッドサーボ回路

- 6 光ピックアップ  
7 スピンドルサーボ回路  
8 データバッファ  
11 データ入力部  
12 演算処理部  
13 記憶部  
14 エラー出力部

【図 1】

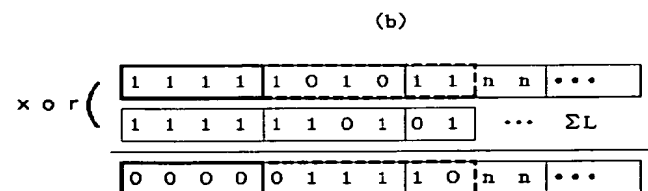
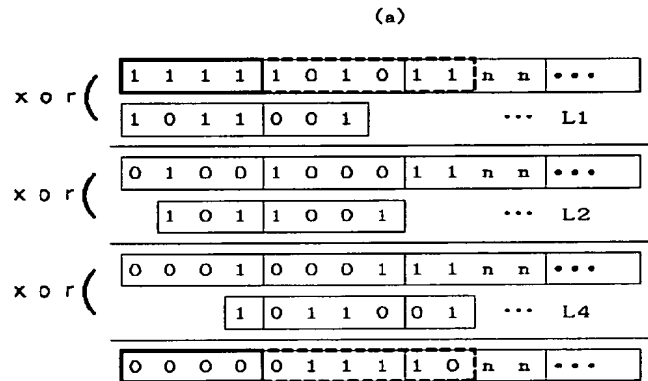
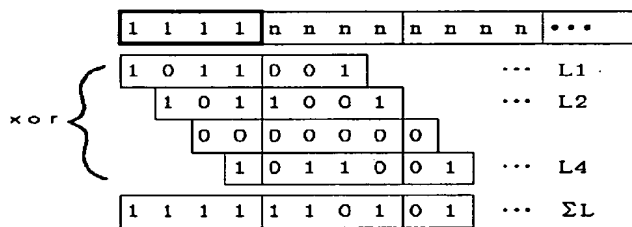
【図 2】



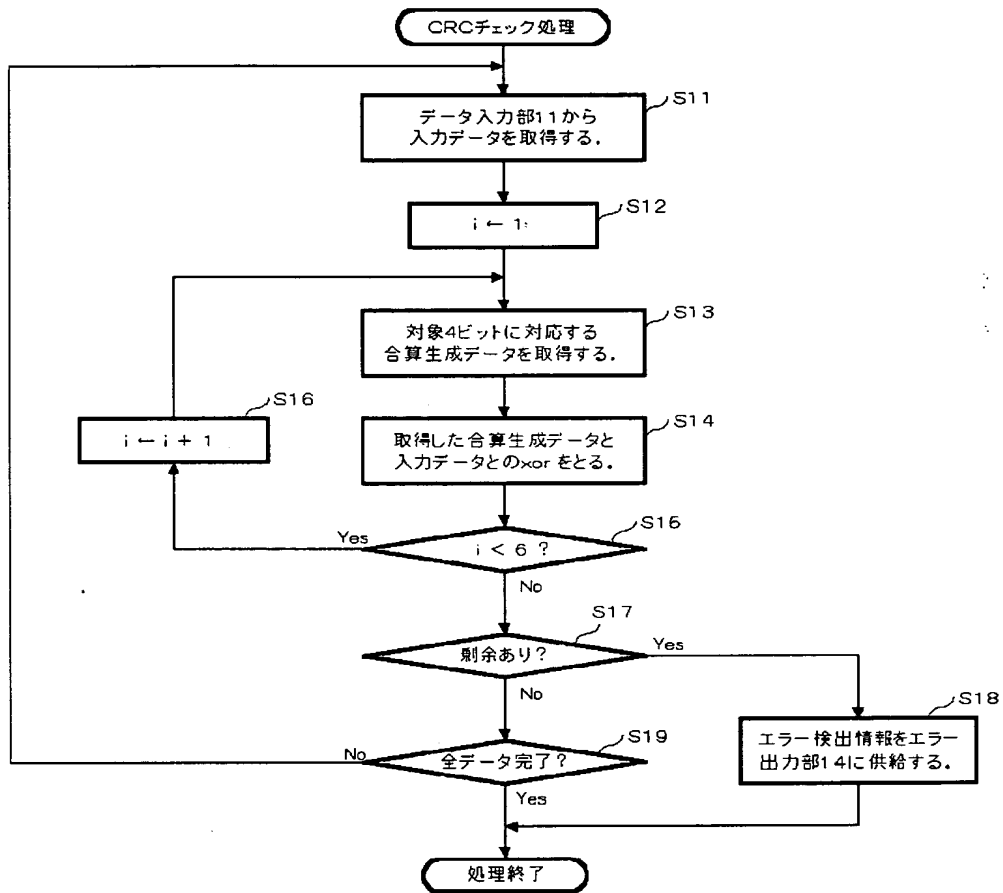
| 対象ビット | 合成生成データ    |
|-------|------------|
| 0000  | 0000000000 |
| 0001  | 0001011001 |
| 0010  | 0010110010 |
| ...   | ...        |
| 1001  | 1001111010 |
| 1010  | 1010010001 |
| ...   | ...        |
| 1100  | 1100011110 |
| ...   | ...        |
| 1110  | 1110101100 |
| 1111  | 111110101  |

【図 3】

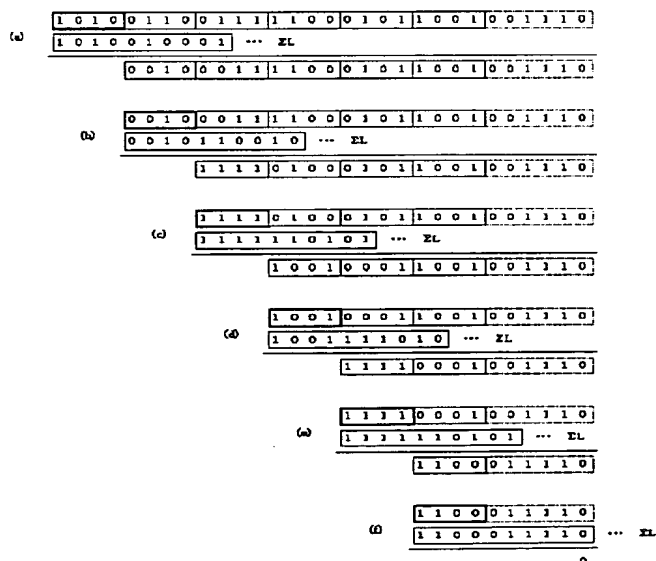
【図 4】



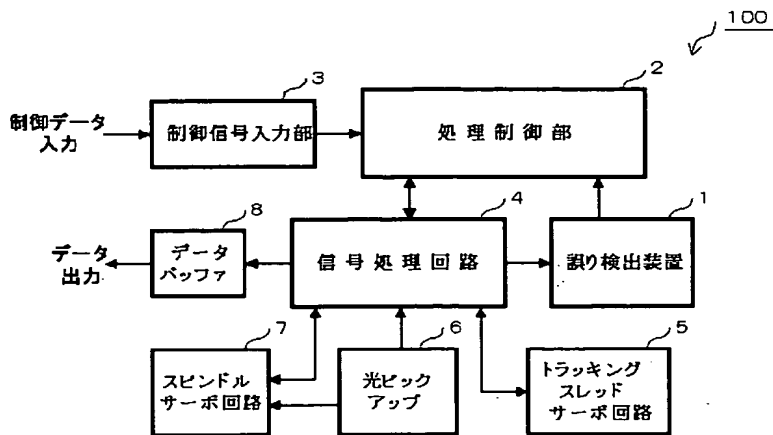
【図 5】



【図 6】



【圖 7】



【图 8】

(a)

[illegible]

(b)

|       | 1 1 1 1 1 1 1 1 | D2059(byte)     | D2058(byte)     | D2057(byte)     | D2056(byte)     | ...    |
|-------|-----------------|-----------------|-----------------|-----------------|-----------------|--------|
| xor { | 1 1 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 1 1 0 0 0 0 0 0 | 0 0 0 0 1 1 0 1 | 1               | ... L1 |
|       | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0             |        |
|       | 1 1 0 0 0 0     | 0 0 0 0 0 0 0 0 | 0 0 1 1 0 0 0 0 | 0 0 0 0 0 0 1 1 | 0 1 1           | ... L3 |
|       | 0 0 0 0 0 0     | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0         |        |
|       | 1 1 0 0         | 0 0 0 0 0 0 0 0 | 0 0 0 0 1 1 0 0 | 0 0 0 0 0 0 0 0 | 1 1 0 1 1       | ... L5 |
|       | 0 0 0           | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0     |        |
|       | 1 1             | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 1 1 | 0 0 0 0 0 0 0 0 | 0 0 1 1 0 1 1   | ... L7 |
|       | 0               | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |        |
|       | 1 1 1 1 1 1 1 1 | 0 0 0 0 0 0 0 0 | 1 1 1 1 1 1 1 1 | 0 0 0 0 1 1 1 0 | 0 0 0 0 1 1 1 0 | ... ΣL |

【図 9】

$$\begin{array}{r}
 x^{23} + x^{20} + x^{17} + x^{15} + x^{14} + x^{12} + x^{11} + x^9 + x^7 + x^6 + x^3 + x^2 + x \\
 x^6 + x^4 + x^3 + 1 \quad \overline{) \quad x^{29} + x^{27} + x^{24} + x^{23} + x^{20} + x^{19} + x^{18} + x^{17} + x^{16} + x^{12} + x^{10} + x^9 + x^6} \\
 \underline{x^{29} + x^{27} + x^{26} + x^{23}} \\
 x^{26} + x^{24} + x^{20} + x^{19} + x^{18} + x^{17} + x^{16} + x^{12} + x^{10} + x^9 + x^6 \\
 \underline{x^{26} + x^{24} + x^{23} + x^{20}} \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 x^7 + x^5 + x^4 + x^3 + x^2 \\
 \underline{x^7 + x^5 + x^4 + x} \\
 x^3 + x^2 + x
 \end{array}$$

【図 11】

$$\begin{array}{r}
 \text{xor} \left( \begin{array}{cccccccccccccccccccccccccccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{cccccccccccccccccccccccccccccccc} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & \end{array} \right) \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 \text{xor} \left( \begin{array}{cccccccc} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & & \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{cccccccc} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & & \end{array} \right) \\
 \hline
 0
 \end{array}$$

【図 10】

$$\begin{array}{r}
 \text{xor} \left( \begin{array}{l} 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 \text{xor} \left( \begin{array}{l} 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \end{array} \right) \\
 \hline
 1\ 1\ 1\ 0
 \end{array}$$